Hackathon

1. Visão Geral do Sistema
2. Arquitetura
3. Documentação da API com Swagger
4. Diagrama de Classes

Visão Geral do Sistema

Desenvolver um sistema eficiente para o processamento de pagamentos de operadoras de cartão de crédito. Nosso objetivo principal é receber os dados das transações com cartão de crédito e validar se o cartão do cliente possui limite disponível para a realização da compra. Este sistema garantirá a verificação precisa e em tempo real do limite de crédito dos clientes, proporcionando uma experiência de pagamento segura e confiável.

Como inicializar a aplicação

Pré-requisito: é necessário ter o Docker instalado na maquina para a execução das instruções a seguir.

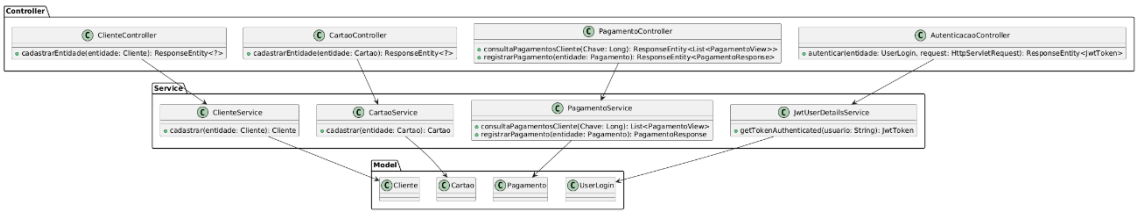
Na Raiz do projeto há um arquivo docker-compose.yaml, em um terminal de comando (CMD) execute o comando “docker-compose build”.

Ao termino da execução da instrução anterior execute o comando “docker-compose up” e aguarte até que todos o serviços sejam inicializados por completo.

O tempo de inicialização da instrução 2 apresentado acima pode variar em decorrência as configurações do host da aplicação portanto é necessário acompanhar os logos para certificar que todos os serviços estejam disponíveis.

Arquitetura

A arquitetura do projeto é baseada em microsserviços, garantindo a autonomia e a comunicação eficiente entre os componentes. Utilizamos o ecossistema Spring para a implementação dos serviços e cada serviço possui uma base de dados isolada.



- Backend: Baseado em MVC.
- Banco de Dados: Utilizamos MYSQL para persistência de dados.

A escolha pela arquitetura MVC neste projeto se justifica por sua capacidade de organizar e modularizar o código de forma eficiente. Cada camada tem um papel específico: o **Model** é responsável pelos dados e pela lógica de negócios, o **View** cuida da interface com o usuário e o **Controller** atua como um intermediário, processando as entradas do usuário e determinando como os dados devem ser apresentados.

No código fornecido, essa separação fica clara:

- **Controllers**: Os controladores (`ClienteController`, `CartaoController`, `PagamentoController`, `AutenticacaoController`) são responsáveis por receber as requisições do usuário, processá-las e devolver uma resposta adequada. Eles chamam os serviços necessários para realizar a lógica de negócios.
- **Services**: As classes de serviço (`ClienteService`, `CartaoService`, `PagamentoService`, `JwtUserDetailsService`) contêm a lógica de negócios e manipulam os dados fornecidos pelos modelos. Elas são invocadas pelos controladores para realizar operações como cadastrar um cliente ou autenticar um usuário.
- **Models**: As entidades (`Cliente`, `Cartao`, `Pagamento`, `UserLogin`) representam os dados do negócio e são manipuladas pelos serviços.

Essas entidades são usadas para mapear e gerenciar as informações persistidas no banco de dados.

Essa estrutura facilita a manutenção e a escalabilidade do sistema. Por exemplo, se for necessário mudar a forma como os dados são apresentados ao usuário, isso pode ser feito na camada de visualização (não mostrada no diagrama) sem alterar a lógica de negócios ou a estrutura dos dados. Da mesma forma, novas funcionalidades podem ser adicionadas ou modificadas na camada de serviço, sem impactar a camada de controle ou de apresentação.

Além disso, a reutilização de componentes é simplificada. O mesmo serviço pode ser utilizado por diferentes controladores, e as mesmas entidades podem ser manipuladas por diversos serviços, economizando tempo e esforço no desenvolvimento. A separação clara das camadas também facilita a criação de testes unitários, permitindo que cada parte do sistema seja testada independentemente, garantindo a integridade e a confiabilidade do código.

Documentação da API com Swagger

Para acessar a documentação detalhada dos endpoints da API, utilize o seguinte link:

http://localhost:8080/api/swagger-ui/index.html#/

Diagrama de Classes